Termination Checking

Simple recursion

```
def append[A](l1: List[A], l2: List[A]): List[A] = l1 match {
    case Cons(x, xs) => Cons(x, append(xs, l2))
    case Nil() => l2
}
```

- Call trace: append(l1, l2), append(l1.tail, l2), append(l1.tail.tail, l2), …
- First argument of append has decreasing size
- Size maps into a well-founded domain

⇒ No infinitely decreasing chain!

Mutual recursion

```
def even(i: BigInt): Boolean = {
  require(i >= 0)
  i == 0 || odd(i - 1)
}
```

```
def odd(i: BigInt): Boolean = {
  require(i >= 0)
  i != 0 && even(i - 1)
}
```

- Call trace: even(i), odd(i - 1), even(i - 2), …
- First argument to even (and odd) always decrease
- BigInt >= 0 is a well-founded domain

⇒ No infinitely decreasing chain!

Lexicographic orderings

```
def lemma[T,U,V](list: List[T], flist: List[U], glist: List[V], f: T => List[U], g: U => List[V]): Boolean = {
  val lhs = append(glist, flatMap(append(flist, flatMap(list, f)), g))
  val rhs = append(append(glist, flatMap(flist, g)), flatMap(list, (x: T) => flatMap(f(x), g)))
  lhs == rhs because (glist match {
    case Cons(ghead, gtail) => lemma(list, flist, gtail, f, g)
    case Nil() => flist match {
      case Cons(fhead, ftail) => lemma(list, ftail, g(fhead), f, g)
      case Nil() => list match {
        case Cons(head, tail) => lemma(tail, f(head), Nil(), f, g)
        case Nil() => true
  }}})
}.holds
```

Recursive calls

Lexicographic orderings

```
def lemma[T,U,V](list: List[T], flist: List[U], glist: List[V], f: T => List[U], g: U => List[V]): Boolean = {
  val lhs = append(glist, flatMap(append(flist, flatMap(list, f)), g))
  val rhs = append(append(glist, flatMap(flist, g)), flatMap(list, (x: T) => flatMap(f(x), g)))
  lhs == rhs because (glist match {
    case Cons(ghead, gtail) => lemma(list, flist, gtail, f, g)
    case Nil() => flist match {
      case Cons(fhead, ftail) => lemma(list, ftail, g(fhead), f, g)
      case Nil() => list match {
        case Cons(head, tail) => lemma(tail, f(head), Nil(), f, g)
        case Nil() => true
  }}})
}.holds
```

tail.size < list.size

Lexicographic orderings

```
def lemma[T,U,V](list: List[T], flist: List[U], glist: List[V], f: T => List[U], g: U => List[V]): Boolean = {
  val lhs = append(glist, flatMap(append(flist, flatMap(list, f)), g))
  val rhs = append(append(glist, flatMap(flist, g)), flatMap(list, (x: T) => flatMap(f(x), g)))
  lhs == rhs because (glist match {
    case Cons(ghead, gtail) => lemma(list, flist, gtail, f, g)
    case Nil() => flist match {
      case Cons(fhead, ftail) => lemma(list, ftail, g(fhead), f, g)
      case Nil() => list match {
        case Cons(head, tail) => lemma(tail, f(head), Nil(), f, g
        case Nil() => true
  }}})
}.holds
```

ftail.size < flist.size

list.size == list.size

Lexicographic orderings

```
def lemma[T,U,V](list: List[T], flist: List[U], glist: List[V], f: T => List[U], g: U => List[V]): Boolean = {
  val lhs = append(glist, flatMap(append(flist, flatMap(list, f)), g))
  val rhs = append(append(glist, flatMap(flist, g)), flatMap(list, (x: T) => flatMap(f(x), g)))
  lhs == rhs because (glist match {
    case Cons(ghead, gtail) => lemma(list, flist, gtail, f, g)
    case Nil() => flist match {
      case Cons(fhead, ftail) => lemma(list, ftail, g(fhead), f, g
      case Nil() => list match {
        case Cons(head, tail) => lemma(tail, f(head), Nil(), f, g
        case Nil() => true
  }}}})
}.holds
```

gtail.size < glist.size

ftail.size == flist.size

list.size == list.size

Decreases construct

```
def M(n: BigInt): BigInt = {

    if (n > 100) n - 10 else M(M(n + 11))
}
```

Decreases construct

```
def M(n: BigInt): BigInt = {
    decreases(101 - n)
    if (n > 100) n - 10 else M(M(n + 11))
} ensuring (_ == (if (n > 100) n - 10 else BigInt(91)))
```